```
d hist

     (FILE 'USPAT' ENTERED AT 08:27:42 ON 29 JAN 1999)
L1          2967 S SPELLING OR SPELL OR SPELLS
L2           939 S UTTERANCE
L3           783 S WORD RECOGNI?
L4            39 S L1 AND L2 AND L3
L5          1868 S VOCABULARY
L6            38 S L4 AND L5
L7        210690 S ADD
L8            26 S L7 (5A) L5
L9             6 S L6 AND L8
L10        91317 S RULES OR RULE
L11          908 S PRONUNCIAT?
L12            2 S L9 AND L10
L13            2 S L12 AND L11
L14           10 S L1 (5A) L2
L15            1 S L12 AND L14
L16            1 S L13 AND L14
L17            1 S L15 AND L16
L18       261754 S MATCH?
L19          629 S CLASSIF? (5A) (WORD OR WORDS)
L20       192866 S LETTER OR LETTERS
L21         1356 S TWO (2W) L20
L22         1061 S PHONEME?
L23            2 S L18 AND L19 AND L21 AND L22
L24            2 S L23 AND L5
L25         3818 S ALPHABETICAL
L26            0 S L25 (5A) L22
L27            1 S L25 (10A) L22
L28          330 S L25 (2A) LIST?
L29            7 S L22 AND L28
L30            0 S L22(10A) L28
L31            0 S L22(20A) L28
L32         8168 S ALPHABETIC?
L33            0 S L22 (5A) L25
L34            1 S L22 (10A) L25

=> s l32 and l22 and l18

L35           71 L32 AND L22 AND L18

=> s l19 and l35

L36            2 L19 AND L35

=> d 1-

1.  5,815,639, Sep. 29, 1998, Computer-aided transcription system using
pronounceable substitute text with a common cross-reference library;
James D. Bennett, et al., 704/235, 270 [IMAGE AVAILABLE]

2.  5,329,609, Jul. 12, 1994, Recognition apparatus with function of
displaying plural recognition candidates; Toru Sanada, et al., 704/251,
235, 276 [IMAGE AVAILABLE]

=> d kwic 2
```

```
d hist

     (FILE 'USPAT' ENTERED AT 08:27:42 ON 29 JAN 1999)
L1          2967 S SPELLING OR SPELL OR SPELLS
L2           939 S UTTERANCE
L3           783 S WORD RECOGNI?
L4            39 S L1 AND L2 AND L3
L5          1868 S VOCABULARY
L6            38 S L4 AND L5
L7        210690 S ADD
L8            26 S L7 (5A) L5
L9             6 S L6 AND L8
L10        91317 S RULES OR RULE
L11          908 S PRONUNCIAT?
L12            2 S L9 AND L10
L13            2 S L12 AND L11
L14           10 S L1 (5A) L2
L15            1 S L12 AND L14
L16            1 S L13 AND L14
L17            1 S L15 AND L16
L18       261754 S MATCH?
L19          629 S CLASSIF? (5A) (WORD OR WORDS)
L20       192866 S LETTER OR LETTERS
L21         1356 S TWO (2W) L20
L22         1061 S PHONEME?
L23            2 S L18 AND L19 AND L21 AND L22
L24            2 S L23 AND L5
L25         3818 S ALPHABETICAL
L26            0 S L25 (5A) L22
L27            1 S L25 (10A) L22
L28          330 S L25 (2A) LIST?
L29            7 S L22 AND L28
L30            0 S L22(10A) L28
L31            0 S L22(20A) L28

=> s l23 1-

MISSING OPERATOR 'L23 1-'

=> d l23 1-

1.  4,977,599, Dec. 11, 1990, Speech recognition employing a set of
Markov models that includes Markov models representing transitions to and
from silence; Lalit R. Bahl, et al., 704/256, 243, 245 [IMAGE AVAILABLE]

2.  4,718,094, Jan. 5, 1988, Speech recognition system; Lalit R. Bahl, et
al., 704/256, 240, 251, 252, 255 [IMAGE AVAILABLE]
```

d hist

```
     (FILE 'USPAT' ENTERED AT 07:53:49 ON 20 JAN 1999)
L1          3028 S SPELL OR SPELLS OR SPELLING OR SPELLINGS
L2          3000 S SPEECH (5A) RECOGNI?
L3           225 S L1 AND L2
L4          1118 S UTTERANCE?
L5           101 S L3 AND L4
L6           903 S PRONUNCIAT?
L7            44 S L5 AND L6
L8           151 S L1 (P) L6
L9            28 S L7 AND L8
                 SET HIGH OFF
L10           44 S L7 AND L7
                 SET HIGH ON
L11           44 S L10 AND L1
L12           44 S L11 AND L6
L13           44 S L12 AND L4
L14         1860 S VOCABULARY
L15           34 S L13 AND L14

=> d l13 1-
```

1. 5,850,627, Dec. 15, 1998, Apparatuses and methods for training and operating speech recognition systems; Joel M. Gould, et al., 704/231, 255, 256 [IMAGE AVAILABLE]

2. 5,832,435, Nov. 3, 1998, Methods for controlling the generation of speech from text representing one or more names; Kim Ernest Alexander Silverman, 704/260, 9, 266 [IMAGE AVAILABLE]

3. 5,819,220, Oct. 6, 1998, Web triggered word set boosting for speech interfaces to the world wide web; Ramesh Sarukkai, et al., 704/243, 240, 270; 706/11 [IMAGE AVAILABLE]

4. 5,806,030, Sep. 8, 1998, Low complexity, high accuracy clustering method for speech recognizer; Jean-Claude Junqua, 704/245, 240, 254, 255, 258 [IMAGE AVAILABLE]

5. 5,799,276, Aug. 25, 1998, Knowledge-based speech recognition system and methods having frame length computed based upon estimated pitch period of vocalic intervals; Edward Komissarchik, et al., 704/251, 207, 208, 231, 257 [IMAGE AVAILABLE]

6. 5,794,189, Aug. 11, 1998, Continuous speech recognition; Joel M. Gould, 704/231, 232, 251, 257, 258 [IMAGE AVAILABLE]

7. 5,774,628, Jun. 30, 1998, Speaker-independent dynamic vocabulary and grammar in speech recognition; Charles T. Hemphill, 704/255, 243, 244, 256, 275 [IMAGE AVAILABLE]

8. 5,758,023, May 26, 1998, Multi-language speech recognition system; Theodore Austin Bordeaux, 704/232, 235 [IMAGE AVAILABLE]

9. 5,751,906, May 12, 1998, Method for synthesizing speech from text and for **spelling** all or portions of the text by analogy; Kim Ernest Alexander Silverman, 704/260, 258, 266 [IMAGE AVAILABLE]

10. 5,749,071, May 5, 1998, Adaptive methods for controlling the annunciation rate of synthesized speech; Kim Ernest Alexander Silverman, 704/260, 258, 266, 267 [IMAGE AVAILABLE]

11. 5,748,840, May 5, 1998, Methods and apparatus for improving the reliability of recognizing words in a large database when the words are spelled or spoken; Charles La Rue, 704/254, 251 [IMAGE AVAILABLE]

12. 5,732,395, Mar. 24, 1998, Methods for controlling the generation of speech from text representing names and addresses; Kim Ernest Alexander Silverman, 704/260, 258, 266, 267 [IMAGE AVAILABLE]

13. 5,727,950, Mar. 17, 1998, Agent based instruction system and method; Donald A. Cook, deceased, et al., 434/350; 345/329, 336, 357, 978 [IMAGE AVAILABLE]

14. 5,724,481, Mar. 3, 1998, Method for automatic speech recognition of arbitrary spoken words; Roger Borgan Garberg, et al., 704/243; 379/88.01; 704/251 [IMAGE AVAILABLE]

15. 5,717,828, Feb. 10, 1998, Speech recognition apparatus and method for learning; Martin Rothenberg, 704/270; 434/185; 704/251 [IMAGE AVAILABLE]

16. 5,682,501, Oct. 28, 1997, Speech synthesis system; Richard Anthony Sharman, 704/260, 256, 257, 258, 261, 266, 269 [IMAGE AVAILABLE]

17. 5,652,828, Jul. 29, 1997, Automated voice synthesis employing enhanced prosodic treatment of text, **spelling** of text and rate of annunciation; Kim Ernest Alexander Silverman, 704/260, 258, 266, 267 [IMAGE AVAILABLE]

18. 5,652,789, Jul. 29, 1997, Network based knowledgeable assistant; Richard A. Miner, et al., 379/201, 88.22, 202 [IMAGE AVAILABLE]

19. 5,638,425, Jun. 10, 1997, Automated directory assistance system using word recognition and phoneme processing method; Frank E. Meador, III, et al., 379/88.01, 88.16, 88.24, 201; 704/231, 236, 251, 270 [IMAGE AVAILABLE]

20. 5,623,578, Apr. 22, 1997, Speech recognition system allows new vocabulary words to be added without requiring spoken samples of the words; Rajendra P. Mikkilineni, 704/255, 232, 240 [IMAGE AVAILABLE]

21. 5,615,299, Mar. 25, 1997, Speech recognition using dynamic features; Lahit R. Bahl, et al., 704/254, 233, 240, 242, 256 [IMAGE AVAILABLE]

22. 5,526,463, Jun. 11, 1996, System for processing a succession of **utterances** spoken in continuous or discrete form; Laurence S. Gillick, et al., 704/251, 231 [IMAGE AVAILABLE]

23. 5,500,920, Mar. 19, 1996, Semantic co-occurrence filtering for speech recognition and signal transcription applications; Julian M. Kupiec, 704/270, 7, 275, 277 [IMAGE AVAILABLE]

24. 5,455,889, Oct. 3, 1995, Labelling speech using context-dependent acoustic prototypes; Lalit R. Bahl, et al., 704/236, 200, 231, 242, 243, 254, 256 [IMAGE AVAILABLE]

25. 5,440,663, Aug. 8, 1995, Computer system for speech recognition; Gerald Moese, et al., 704/255, 200, 251, 256 [IMAGE AVAILABLE]

26. 5,428,707, Jun. 27, 1995, Apparatus and methods for training speech recognition systems and their users and otherwise improving speech

recognition performance; Joel M. Gould, et al., 704/231, 251, 255 [IMAGE AVAILABLE]

27. 5,369,726, Nov. 29, 1994, Speech recognition circuitry employing nonlinear processing speech element modeling and phoneme estimation; John P. Kroeker, et al., 704/236 [IMAGE AVAILABLE]

28. 5,293,584, Mar. 8, 1994, Speech recognition system for natural language translation; Peter F. Brown, et al., 704/277, 200, 257, 270 [IMAGE AVAILABLE]

29. 5,293,451, Mar. 8, 1994, Method and apparatus for generating models of spoken words based on a small number of **utterances**; Peter F. Brown, et al., 704/245 [IMAGE AVAILABLE]

30. 5,283,833, Feb. 1, 1994, Method and apparatus for speech processing using morphology and rhyming; Kenneth W. Church, et al., 704/252; 379/52, 88.01, 88.14, 88.16 [IMAGE AVAILABLE]

31. 5,267,345, Nov. 30, 1993, Speech recognition apparatus which predicts word classes from context and words from word classes; Peter F. Brown, et al., 704/255 [IMAGE AVAILABLE]

32. 5,222,188, Jun. 22, 1993, Method and apparatus for speech recognition based on subsyllable **spellings**; Sandra E. Hutchins, 704/200 [IMAGE AVAILABLE]

33. 5,208,897, May 4, 1993, Method and apparatus for speech recognition based on subsyllable **spellings**; Sandra E. Hutchins, 704/200 [IMAGE AVAILABLE]

34. 5,202,952, Apr. 13, 1993, Large-vocabulary continuous speech prefiltering and processing system; Laurence S. Gillick, et al., 704/200 [IMAGE AVAILABLE]

35. 5,182,773, Jan. 26, 1993, Speaker-independent label coding apparatus; Lalit R. Bahl, et al., 704/222 [IMAGE AVAILABLE]

36. 5,177,685, Jan. 5, 1993, Automobile navigation system using real time spoken driving instructions; James R. Davis, et al., 701/200; 340/988; 701/209, 211, 220 [IMAGE AVAILABLE]

37. 5,170,432, Dec. 8, 1992, Method of speaker adaptive speech recognition; Heidi Hackbarth, et al., 704/254 [IMAGE AVAILABLE]

38. 5,168,524, Dec. 1, 1992, Speech-recognition circuitry employing nonlinear processing, speech element modeling and phoneme estimation; John P. Kroeker, et al., 704/254, 231 [IMAGE AVAILABLE]

39. 5,091,950, Feb. 25, 1992, Arabic language translating device with **pronunciation** capability using language **pronunciation** rules; Moustafa E. Ahmed, 704/277, 3, 7 [IMAGE AVAILABLE]

40. 5,072,452, Dec. 10, 1991, Automatic determination of labels and Markov word models in a speech recognition system; Peter F. Brown, et al., 704/256 [IMAGE AVAILABLE]

41. 5,054,074, Oct. 1, 1991, Optimized speech recognition system and method; Raimo Bakis, 704/240 [IMAGE AVAILABLE]

42. 5,027,406, Jun. 25, 1991, Method for interactive speech recognition and training; Jed Roberts, et al., 704/244, 251 [IMAGE AVAILABLE]

43. 4,884,972, Dec. 5, 1989, Speech synchronized animation; Elon Gasper, 434/185; 345/302, 473; 434/167, 169, 307R; 704/235, 276 [IMAGE AVAILABLE]

44. 4,741,036, Apr. [redacted] 1988, Determination of phone [redacted]ghts for markov models in a speech recognition system; Lalit R. Bahl, et al., 704/256 [IMAGE AVAILABLE]

=> d l15 1-

1. 5,850,627, Dec. 15, 1998, Apparatuses and methods for training and operating speech recognition systems; Joel M. Gould, et al., 704/231, 255, 256 [IMAGE AVAILABLE]

2. 5,819,220, Oct. 6, 1998, Web triggered word set boosting for speech interfaces to the world wide web; Ramesh Sarukkai, et al., 704/243, 240, 270; 706/11 [IMAGE AVAILABLE]

3. 5,799,276, Aug. 25, 1998, Knowledge-based speech recognition system and methods having frame length computed based upon estimated pitch period of vocalic intervals; Edward Komissarchik, et al., 704/251, 207, 208, 231, 257 [IMAGE AVAILABLE]

4. 5,794,189, Aug. 11, 1998, Continuous speech recognition; Joel M. Gould, 704/231, 232, 251, 257, 258 [IMAGE AVAILABLE]

5. 5,774,628, Jun. 30, 1998, Speaker-independent dynamic **vocabulary** and grammar in speech recognition; Charles T. Hemphill, 704/255, 243, 244, 256, 275 [IMAGE AVAILABLE]

6. 5,758,023, May 26, 1998, Multi-language speech recognition system; Theodore Austin Bordeaux, 704/232, 235 [IMAGE AVAILABLE]

7. 5,748,840, May 5, 1998, Methods and apparatus for improving the reliability of recognizing words in a large database when the words are spelled or spoken; Charles La Rue, 704/254, 251 [IMAGE AVAILABLE]

8. 5,727,950, Mar. 17, 1998, Agent based instruction system and method; Donald A. Cook, deceased, et al., 434/350; 345/329, 336, 357, 978 [IMAGE AVAILABLE]

9. 5,724,481, Mar. 3, 1998, Method for automatic speech recognition of arbitrary spoken words; Roger Borgan Garberg, et al., 704/243; 379/88.01; 704/251 [IMAGE AVAILABLE]

10. 5,717,828, Feb. 10, 1998, Speech recognition apparatus and method for learning; Martin Rothenberg, 704/270; 434/185; 704/251 [IMAGE AVAILABLE]

11. 5,682,501, Oct. 28, 1997, Speech synthesis system; Richard Anthony Sharman, 704/260, 256, 257, 258, 261, 266, 269 [IMAGE AVAILABLE]

12. 5,652,789, Jul. 29, 1997, Network based knowledgeable assistant; Richard A. Miner, et al., 379/201, 88.22, 202 [IMAGE AVAILABLE]

13. 5,638,425, Jun. 10, 1997, Automated directory assistance system using word recognition and phoneme processing method; Frank E. Meador, III, et al., 379/88.01, 88.16, 88.24, 201; 704/231, 236, 251, 270 [IMAGE AVAILABLE]

14. 5,623,578, Apr. 22, 1997, Speech recognition system allows new **vocabulary** words to be added without requiring spoken samples of the words; Rajendra P. Mikkilineni, 704/255, 232, 240 [IMAGE AVAILABLE]

15. 5,615,299, Mar. 25, 1997, Speech recognition using dynamic features; Lahit R. Bahl, et al., 704/254, 233, 240, 242, 256 [IMAGE AVAILABLE]

16. 5,526,463, Jun. 11, 1996, System for processing a succession of **utterances** spoken in continuous or discrete form; Laurence S. Gillick, et al., 704/251, 231 [IMAGE AVAILABLE]

17. 5,500,920, Mar. 19, 1996, Semantic co-occurrence filtering for speech recognition and signal transcription applications; Julian M. Kupiec, 704/270, 7, 275, 277 [IMAGE AVAILABLE]

18. 5,455,889, Oct. 3, 1995, Labelling speech using context-dependent acoustic prototypes; Lalit R. Bahl, et al., 704/236, 200, 231, 242, 243, 254, 256 [IMAGE AVAILABLE]

19. 5,428,707, Jun. 27, 1995, Apparatus and methods for training speech recognition systems and their users and otherwise improving speech recognition performance; Joel M. Gould, et al., 704/231, 251, 255 [IMAGE AVAILABLE]

20. 5,293,584, Mar. 8, 1994, Speech recognition system for natural language translation; Peter F. Brown, et al., 704/277, 200, 257, 270 [IMAGE AVAILABLE]

21. 5,293,451, Mar. 8, 1994, Method and apparatus for generating models of spoken words based on a small number of **utterances**; Peter F. Brown, et al., 704/245 [IMAGE AVAILABLE]

22. 5,267,345, Nov. 30, 1993, Speech recognition apparatus which predicts word classes from context and words from word classes; Peter F. Brown, et al., 704/255 [IMAGE AVAILABLE]

23. 5,222,188, Jun. 22, 1993, Method and apparatus for speech recognition based on subsyllable **spellings**; Sandra E. Hutchins, 704/200 [IMAGE AVAILABLE]

24. 5,208,897, May 4, 1993, Method and apparatus for speech recognition based on subsyllable **spellings**; Sandra E. Hutchins, 704/200 [IMAGE AVAILABLE]

25. 5,202,952, Apr. 13, 1993, Large-**vocabulary** continuous speech prefiltering and processing system; Laurence S. Gillick, et al., 704/200 [IMAGE AVAILABLE]

26. 5,182,773, Jan. 26, 1993, Speaker-independent label coding apparatus; Lalit R. Bahl, et al., 704/222 [IMAGE AVAILABLE]

27. 5,177,685, Jan. 5, 1993, Automobile navigation system using real time spoken driving instructions; James R. Davis, et al., 701/200; 340/988; 701/209, 211, 220 [IMAGE AVAILABLE]

28. 5,170,432, Dec. 8, 1992, Method of speaker adaptive speech recognition; Heidi Hackbarth, et al., 704/254 [IMAGE AVAILABLE]

29. 5,091,950, Feb. 25, 1992, Arabic language translating device with **pronunciation** capability using language **pronunciation** rules; Moustafa E. Ahmed, 704/277, 3, 7 [IMAGE AVAILABLE]

30. 5,072,452, Dec. 10, 1991, Automatic determination of labels and Markov word models in a speech recognition system; Peter F. Brown, et al., 704/256 [IMAGE AVAILABLE]

31. 5,054,074, Oct. 1, 1991, Optimized speech recognition system and method; Raimo Bakis, 704/240 [IMAGE AVAILABLE]

32. 5,027,406, Jun. 25, 1991, Method for interactive speech recognition and training; Jed Roberts, et al., 704/244, 251 [IMAGE AVAILABLE]

33. 4,884,972, Dec. 5, 1989, Speech synchronized animation; Elon Gasper, 434/185; 345/302, 473; 4/167, 169, 307R; 704/235, 27[redacted]IMAGE AVAILABLE]

34. 4,741,036, Apr. 26, 1988, Determination of phone weights for markov models in a speech recognition system; Lalit R. Bahl, et al., 704/256 [IMAGE AVAILABLE]

d hist

```
      (FILE 'USPAT' ENTERED AT 07:53:49 ON 20 JAN 1999)
L1          3028 S SPELL OR SPELLS OR SPELLING OR SPELLINGS
L2          3000 S SPEECH (5A) RECOGNI?
L3           225 S L1 AND L2
L4          1118 S UTTERANCE?
L5           101 S L3 AND L4
L6           903 S PRONUNCIAT?
L7            44 S L5 AND L6
L8           151 S L1 (P) L6
L9            28 S L7 AND L8
                 SET HIGH OFF
L10           44 S L7 AND L7
                 SET HIGH ON
L11           44 S L10 AND L1
L12           44 S L11 AND L6
L13           44 S L12 AND L4
L14         1860 S VOCABULARY
L15           34 S L13 AND L14
L16          102 S (PHONETIC OR PHONEMIC) (5A) L1
L17           13 S L15 AND L16
L18           72 S L1 (5A) L6
L19            4 S L17 AND L18

=> d 1-
```

1. 5,850,627, Dec. 15, 1998, Apparatuses and methods for training and operating speech recognition systems; Joel M. Gould, et al., 704/231, 255, 256 [IMAGE AVAILABLE]

2. 5,500,920, Mar. 19, 1996, Semantic co-occurrence filtering for speech recognition and signal transcription applications; Julian M. Kupiec, 704/270, 7, 275, 277 [IMAGE AVAILABLE]

3. 5,222,188, Jun. 22, 1993, Method and apparatus for speech recognition based on subsyllable **spellings**; Sandra E. Hutchins, 704/200 [IMAGE AVAILABLE]

4. 5,208,897, May 4, 1993, Method and apparatus for speech recognition based on subsyllable **spellings**; Sandra E. Hutchins, 704/200 [IMAGE AVAILABLE]

d hist

(FILE 'USPAT' ENTERED AT 08:27:42 ON 29 JAN 1999)

L1	2967	S	SPELLING OR SPELL OR SPELLS
L2	939	S	UTTERANCE
L3	783	S	WORD RECOGNI?
L4	39	S	L1 AND L2 AND L3
L5	1868	S	VOCABULARY
L6	38	S	L4 AND L5
L7	210690	S	ADD
L8	26	S	L7 (5A) L5
L9	6	S	L6 AND L8
L10	91317	S	RULES OR RULE
L11	908	S	PRONUNCIAT?
L12	2	S	L9 AND L10
L13	2	S	L12 AND L11
L14	10	S	L1 (5A) L2
L15	1	S	L12 AND L14
L16	1	S	L13 AND L14
L17	1	S	L15 AND L16

=> d 19 1-

1. 5,850,627, Dec. 15, 1998, Apparatuses and methods for training and operating speech recognition systems; Joel M. Gould, et al., 704/231, 255, 256 [IMAGE AVAILABLE]

2. 5,794,189, Aug. 11, 1998, Continuous speech recognition; Joel M. Gould, 704/231, 232, 251, 257, 258 [IMAGE AVAILABLE]

3. 5,765,132, Jun. 9, 1998, Building speech models for new words in a multi-word **utterance**; Jed M. Roberts, 704/254, 243, 253, 270 [IMAGE AVAILABLE]

4. 5,027,406, Jun. 25, 1991, Method for interactive speech recognition and training; Jed Roberts, et al., 704/244, 251 [IMAGE AVAILABLE]

5. 4,989,248, Jan. 29, 1991, Speaker-dependent connected speech **word recognition** method; Thomas B. Schalk, et al., 704/252 [IMAGE AVAILABLE]

6. 4,831,551, May 16, 1989, Speaker-dependent connected speech **word recognizer**; Thomas B. Schalk, et al., 704/233, 241 [IMAGE AVAILABLE]

=> d l13 1-

1. 5,794,189, Aug. 11, 1998, Continuous speech recognition; Joel M. Gould, 704/231, 232, 251, 257, 258 [IMAGE AVAILABLE]

2. 5,027,406, Jun. 25, 1991, Method for interactive speech recognition and training; Jed Roberts, et al., 704/244, 251 [IMAGE AVAILABLE]

```
/*////////////////////////////////////////////////////////////////////////
//  FILE:                phnspell.cpp
//  CREATED:       2-Jan-96
//  AUTHOR:              Charles Ingold
//  DESCRIPTION:         Pron spelling and frequency table class.
//
//    Copyright (C) Dragon Systems, 1995-1996
//    DRAGON SYSTEMS CONFIDENTIAL
//
// Revision history log
    VSS revision history.  Do not edit by hand.
    $Log: /pq/prons/phnspell.h $

    1     3/24/97 16:30 Chuck
    PHONEQUERY Ver 0.01.165
    Added prons lib
    $NoKeywords: $
*////////////////////////////////////////////////////////////////////////
/


#ifndef _phnspell_h_
#define _phnspell_h_


//#include "trec.h"

//#include "sdapi.h"
//#include "parts.h"

/* PhnSpellArray

To generate a pronunciation for a word, we build a network of rules,
states and words corresponding to phonetic/spelling fragments.  These
phonetic/spelling fragments look like this in an ASCII file:


      a ) 2602     The first column contains spelling fragments,
      a , 753            the second column contains prons for the spelling
fragments,
      a / 3377     and the third column contains the frequency for the given
      a 6 880            spelling/pron pair.
      aa ) 2
      ae , 4             Note: This sample omits a lot of the pron and freq
entries for
      ae / 57            the spellings for the sake of brevity


We store the phonetic transcriptions in a block of zero-terminated strings:.

     29 00 2C 00 2F 00 36 00 40  ).,./.6.@
     00 41 00 45 00 49 00 61 00  .A.E.I.a.
     63 00 65 00 69 00 6F 00 75  c.e.i.o.u
     00 7B 00 50 00 56 00 00 00  .{.P.V...

We store the set of phoneme/spelling entries in a table as follows:
      1) At the offset for a particular string we store the spelling fragment
         as a zero-terminated wide-character Unicode string.
      2) Following the spelling, we store each pron and frequency for that
string
         as an uns16 offset into the block of phonetic transcriptions and an
uns16
```

```
         for the frequency.
      3) We terminate the list of prons and frequencies with the sentinel
uns16 0xffff.

 FFFF  FFFF  0061  0000  0000  0A2A   // Unicode "a", Null, pronOffset 0, and
freq 2602
 0002  02F1  0004  0D31  0006  0370   // pronOffset 2, freq 753, pronOffset 4,
freq 3377
 0008  1A1A  000A  231A  000C  0D31
 000E  01A7  0010  2B62  0012  03FD
 0014  03FD  0016  0370  0018  0204
 001A  1A1A  001C  065A  FFFF  0061   // col 5 terminates "a" entry, 6 starts
"aa\0"
 0061  0000  0000  0002  FFFF  0061   // only one pron for "aa", pronOffset 0,
freq 2
 0065  0000  0002  0004  0004  0039   // Now we have "ae"... with prons
similar to "a"
 0008  0009  000A  0008  000C  0039   // but with different frequencies.
 000E  0024  0010  0001  0012  0008
 0014  0008  001A  0009  FFFF  0061


This PhnSpellArray must be kept in INCREASING ALPHABETIC ORDER ON THE SPELLING
fragments because it is accessed via a hash function which returns the offset
at
which to start searching for spelling fragments which match the initial
characters
of a given string.  The search terminates when a spelling is found which is
alphabetically greater than the target string.

Sizes:
      an entry in the PhnSpell table is
            number of bytes in wide-character string + 2 + (4 * number of
prons) + 2
      an entry in the phoneme table is
            number of bytes in the pron + 2 for the wide terminator.
*/

#define _UNICODE

#define BAD_CHAR_INDEX -1
typedef DgnAC< SDRuleItem > RuleItemArray;

typedef uns16 PhnSpellOffset; // location in PhnSpellDataTable.
typedef uns16 PronOffset;           // location in PronTable.

typedef uns16 PhnSpell;

////////////////////////////////////////////////////////////////////////////////

//
// PronOffsetEntry contains a phonetic transcription and its offset
// in PronTable.  We use a temporary DgnAC<> to avoid duplicates in PronTable,
// but only during readAscii().
//
class PronOffsetEntry {
 public:
      char *mpStr;                  // simply a 0-terminated pron string
      PronOffset mOffset;     // the offset in for mpStr in the pron data.

      PronOffsetEntry() {
            mpStr = 0;
```

```
            mOffset = 0;
        }
        void init(char *pData, PronOffset offset) {
            assert(pData);
            int dataLen = strlen(pData+offset)+1;
            mpStr = DgnNew(char [dataLen] );
            strcpy(mpStr, pData+offset);
            mOffset = offset;
        }

        ~PronOffsetEntry() {
            DgnDeleteArray(mpStr);
        }
};

////////////////////////////////////////////////////////
//
// PronOffsetTbl is used by readAscii() to keep track of the offsets for
// phonetic transcriptions which we have seen before.  It is temporary, and
// only used by readAscii().
//
class PronOffsetTbl : DgnOC<PronOffsetEntry> {
 protected:
        char *mpDataBlock;
        PronOffset mnDataSize;
        PronOffset mCurrentOffset;

 public:
        PronOffsetTbl() {
            mpDataBlock = NULL;            // the current block of 0-terminated
prons
            mnDataSize = 0;
            mCurrentOffset = 0;
        }
        ~PronOffsetTbl() {
            DgnDeleteArray(mpDataBlock);
        }
        PronOffset getOffset(char *pData);
        PronOffset getCurrentOffset() { return mCurrentOffset; }
        char *getCopyOfDataBlock() {
            char *pCopy = DgnNew( char[mCurrentOffset] );
            memcpy(pCopy, mpDataBlock, mCurrentOffset);
            return pCopy;
        }
};

////////////////////////////////////////////////////////
//
// WideCharOffset is used to keep track of the offsets for
// single char spellings in incr alpha order.  We use this to
// look up the first PhnSpell entry which is a partial match for a
// target word.
//
class WideCharOffset {
 protected:
        wchar_t mWChar;
        PhnSpellOffset mPhnSpellOffset;
 public:
        wchar_t *getChar() {return &mWChar;}
        PhnSpellOffset getOffset() {return mPhnSpellOffset;}
        WideCharOffset(wchar_t wChar, PhnSpellOffset offset) {
            mWChar = wChar;
```

```
            mPhnSpellOffset = offset;
        }
};

typedef DgnAC<WideCharOffset> WideCharOffsetTable;

// comparison func for WideCharOffsetTable
int WideCharOffsetCmp(const void *given, const void *test);

#define PHNSPELL_END_OF_ENTRY 0xffff
#define UNS16TOWC(x) (wchar_t) x
#define WCTOUNS16(x) (uns16) x

#define UNS16PTOWCP(x) (wchar_t *) x
#define WCPTOUNS16P(x) (uns16 *) x

class PhnSpellArray  {
// Data
 protected:
      char                    *mpPronData;                    // The pron table

      PronOffset               mnPronDataSize;

      uns16                         *mpPhnSpellData;         // The
Spell/PronOffset/freq table
      PhnSpellOffset     mPhnSpellDataSize;

      WideCharOffsetTable  mWCOffsetTable;             // Offsets for single
char spellings

      wchar_t                 *mpWCTargetSpell;        // The current Target
Spelling
      uns16                         mnWCTargetSpellSize;

    uns32                      mTotalFrequency;
      uns32                    mnEntries;

// Functions
 public:

      PhnSpellArray() {
            mpPronData = NULL;
            mnPronDataSize = 0;
            mpPhnSpellData = NULL;
            mPhnSpellDataSize = 0;
            mpWCTargetSpell = NULL;
            mnWCTargetSpellSize = 0;
            mTotalFrequency = 0;
            mnEntries = 0;
      }

      ~PhnSpellArray() {
            DgnDeleteArray(mpPronData);
            DgnDeleteArray(mpPhnSpellData);
            DgnDeleteArray(mpWCTargetSpell);
      }

    void readAscii(FILE *pDataFile);
      void readBinaryFile(FILE *pDataFile);
      void writeBinaryFile(FILE *pDataFile);

    void PhnSpellArray::getGuessStates(SDhVoc hScratchVoc,
```

```
                                        SDhState hScratchParentState,
                                        SDhState *phStateArray,
                                        const char *szSpelling);

     SDhRule PhnSpellArray::getGuessRule(SDhVoc hScratchVoc,
                                                    SDhState hParentState,
                                                        const char
*szRuleName,
                                                        const char
*szSpelling);

     void PhnSpellArray::getGuessWords(SDhVoc hScratchVoc,
               //                                SDhState
hScratchParentState,
                                                  SDhState hGuessState,
                                                  char *szGuessStateName,
                                                    SDInteger *pTotalFreq,
                                                    PhnSpell *pPhnSpell);

 protected:

     inline int ckPhnSpellPtr( PhnSpell *pPhnSpell )
     {
        return ( pPhnSpell >= mpPhnSpellData  &&
                 pPhnSpell <= mpPhnSpellData + mPhnSpellDataSize );
     }

     inline int ckPronOffset( PronOffset pronOffset )
     {
           return (pronOffset == PHNSPELL_END_OF_ENTRY ||
                       pronOffset <= mnPronDataSize );
     }

     inline int ckPronOffsetPtr( PronOffset *pPronOffset )
     {
        return ( pPronOffset >= mpPhnSpellData  &&
                 pPronOffset <= mpPhnSpellData + mPhnSpellDataSize &&
                 ckPronOffset (*pPronOffset) );
     }

     // find partial match for target spelling and set mpWCTargetSpell
     PhnSpell *firstPhnSpellMatch(const char *pTargetSpell );

     // find partial match for target Unicode spelling
     PhnSpell *firstPhnSpellMatch( wchar_t *pWCTargetSpell );

     // get Offset for next entry Matching mpWCTargetSpell
     PhnSpell *nextPhnSpellMatch( PhnSpell * pPhnSpell );

     // get Offset for next spelling, only called by nextPhnSpellMatch()
     PhnSpell *nextSpelling( PhnSpell *pPhnSpell );

     // get first pronOffset for a spelling
     PronOffset *getOffsetPron0( PhnSpell *pPhnSpell );

     // get next pronOffset for a spelling.
     PronOffset *getOffsetNextPron( PronOffset *pPronOffset );

     char *getPron( PronOffset *pPronOffset );

     // get the frequency for the current pron with the current spelling.
```

```
      SDInteger getFrequency( PronOffset *pPronOffset );

 private:
    PhnSpellArray(const PhnSpellArray&);
    PhnSpellArray& operator=(const PhnSpellArray&);
};

/* old Trec history follows:

    7     10/07/96 11:39 Chuck
    TAHITI Ver 0.04.337

    6     8/06/96 6:24p Chuck
    TAHITI Ver 0.04.252
     We now use pointers instead of offsets when reading PhnSpellArray.
     Less work, easier to read, and we have assertions too.

    5     7/29/96 10:09a Joel
    TAHITI Ver 0.04.232

    4     7/17/96 2:32p Chuck
    TAHITI Ver 0.04.210
     Removed data members used for bookkeeping purposes, that stuff belongs
    to
     the caller now, which is in phnguess.{h,cpp}.

    3     7/10/96 3:39p Chuck
    TAHITI Ver 0.04.198

    2     7/08/96 8:10p Chuck
    TAHITI Ver 0.04.194

    1     5/18/96 11:01p Tim
    Moving over from TLIB.
    $NoKeywords: $

     Old TLIB revision history follows.
     *tlib-revision-history*
    1 phnspell.h 02-Feb-96,18:00:50,'CHUCK' TAHITI Ver 0.03.222
    2 phnspell.h 14-Mar-96,11:12:42,'CHUCK' TAHITI Ver 0.03.321
    3 phnspell.h 27-Mar-96,10:47:00,'CHUCK' TAHITI Ver 0.03.350
    4 phnspell.h 01-Apr-96,12:56:44,'CHUCK' TAHITI Ver 0.03.363
    5 phnspell.h 08-Apr-96,09:18:10,'CHUCK' TAHITI Ver 0.03.375    6
phnspell.h 17-May-96,20:03:48,'CHUCK' TAHITI Ver 0.04.097
    7 PHNSPELL.H 18-May-96,18:54:52,'TIM' TAHITI Ver 0.04.100
     *tlib-revision-history*

     Revision 7 on Sat May 18 18:54:36 1996 by tim TAHITI Ver 0.04.100

     Revision 6 on Fri May 17 20:03:46 1996 by Chuck TAHITI Ver 0.04.097
       Redesigning iterface...

     Revision 5 on Mon Apr 08 09:18:08 1996 by Chuck TAHITI Ver 0.03.375
       Support for persistant PhnSpellArray object

     Revision 4 on Mon Apr 01 12:56:42 1996 by Chuck TAHITI Ver 0.03.363
       Support for gudtest and instrumentation for built/dict words

     Revision 3 on Wed Mar 27 10:46:58 1996 by Chuck TAHITI Ver 0.03.350
       Restructured code to get rid of static sizes.

     Revision 2 on Thu Mar 14 11:12:40 1996 by Chuck TAHITI Ver 0.03.321
```

```
    Revision 1 on Fri Feb 02 18:00:48 1996 by Chuck TAHITI Ver 0.03.222
//
//////////////////////////////////////////////////////////////////////////////
*/
#endif
```

```
/* ////////////////////////////////////////////////////////////////////////
//   FILE:              prnguessr.cpp
//  CREATED:       2-Feb-96
//  AUTHOR:              Chuck Ingold
//  DESCRIPTION:         Apputil level pron guesser.
//
//    Copyright (C) Dragon Systems, 1995-1996
//    DRAGON SYSTEMS CONFIDENTIAL
//
// Revision history log
    VSS revision history.  Do not edit by hand.
    $Log: /pq/prons/prnguess.cpp $

    1     3/24/97 16:30 Chuck
    PHONEQUERY Ver 0.01.165
    Added prons lib
    $NoKeywords: $
*/ ////////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "phnspell.h"

#if 0
//#include "trec.h"
#include "myassert.h"
#include "cutil.h"
#include "assert.h"
//#include "apputil.h"
#include "phnspell.h"
#include "ckapi.h"
#include "chlist.h"
#include "prnguess.h"
#include "dump.h"
#endif
DEF_ERR( PronGuesser, 1, "PronGuesser is uninitialized" );
DEF_ERR( PronGuesser, 2, "Invalid handle argument %d for %s" ); // %d handle
%s argument name
DEF_ERR( PronGuesser, 3, "NULL Pointer argument for %s" );  // %s argument
name
DEF_ERR( PronGuesser, 4, "No pron available for '%s' when hPronResult == 0" );
// %s argument name

PhnSpellArray *spPhnSpellArray =0;

////////////////////////////////////////////////////////////////////////
//
// PronGuesser_LoadAscii()
//
//   Initializes the internal PronGuesser data from an ascii file.
//
void PronGuesser_LoadAscii(const char *szFileName)
{
      FILE *pDataFile = fopen( szFileName, "r" );
//    if( ! pDataFile )
//          errThrow( USE_ERR( Global, 2 ), pDataFile );

      xprintf( "PhnSpellDataFile = %s\n", szFileName );
      if (spPhnSpellArray)
            DgnDelete(spPhnSpellArray);
      spPhnSpellArray = DgnNew(PhnSpellArray);

      spPhnSpellArray->readAscii(pDataFile);
```

```
      fclose(pDataFile);
//    memStats( "PhnSpellArray file loaded");
}

////////////////////////////////////////////////////////////////////////
//
// PronGuesser_Load()
//
//   Initializes the internal PronGuesser data from a binary file.
//
void PronGuesser_Load(FILE *pFile)
{
      if (spPhnSpellArray)
            DgnDelete(spPhnSpellArray);
      spPhnSpellArray = DgnNew(PhnSpellArray);
      spPhnSpellArray->readBinaryFile(pFile);
}


////////////////////////////////////////////////////////////////////////
//
// PronGuesser_Save
//
// Writes PronGuesser internal data to a binary file.
//
// FUTURE errThrow if PronGuesser not initialized.
void PronGuesser_Save(FILE *pFile)
{
      if ( !spPhnSpellArray)
      errThrow( USE_ERR( PronGuesser, 1 ) );

      spPhnSpellArray->writeBinaryFile(pFile);
}


////////////////////////////////////////////////////////////////////////
//
// PronGuesser_Terminate()
//
// PronGuesser_Terminate deletes internal PronGuesser data.
void PronGuesser_Terminate()
{
      if ( !spPhnSpellArray)
      errThrow( USE_ERR( PronGuesser, 1 ) );

      DgnDelete(spPhnSpellArray);
}

////////////////////////////////////////////////////////////////////////
//
// PronGuesser_GetRuleFromString
//
// Returns an SDhRule named pRuleName which contains a pron-network
// for szSpelling.
//
// hScratchVoc and hScratchState specify where to create the network of rules,
// states and words for guessing pronunciations.
//
// FUTURE errThrow if PronGuesser not initialized
SDhRule PronGuesser_GetRuleFromString(SDhVoc hScratchVoc,
                                      SDhState hScratchState,
                                const char *szRuleName,
```

```
                                        const char *szSpelling)
{
      if ( !spPhnSpellArray )
      errThrow( USE_ERR( PronGuesser, 1 ) );

      if ( !hScratchVoc )
      errThrow( USE_ERR( PronGuesser, 2 ), hScratchVoc );

      if ( !hScratchVoc )
      errThrow( USE_ERR( PronGuesser, 3 ), szSpelling );

      char *pSpace = strchr( szSpelling, 0x20);

      if ( pSpace == NULL ) {
            return spPhnSpellArray->getGuessRule(hScratchVoc, hScratchState,
                                                                  szRuleName,
szSpelling);

      } else { /// Build a network for each space-delimited token

            int nSpellLen= strlen(szSpelling);
            int nTokens = 0;
          SDhRule *phRules= DgnNewArray( SDhRule, nSpellLen );
            memset(phRules, 0, nSpellLen * sizeof(SDhRule));
          char **pTokenPtrs= DgnNewArray( char *, nSpellLen );
            memset(pTokenPtrs, 0, nSpellLen * sizeof(char *));

            /// figure out number of tokens
          char *pPhrase = DgnNewArray( char, nSpellLen + 1);
            strncpy(pPhrase, szSpelling, nSpellLen + 1);
            char *pWord = pPhrase;

            while (pSpace) {
                  while ( *pWord && *pWord == ' ') {
                        pWord++;
                  }
                  if (*pWord == 0x0) {
                        break;
                  }
                  pSpace = strchr( pWord, 0x20 );
                  if (pSpace) {
                        *pSpace = 0x0;
                  }
                  pTokenPtrs[ nTokens++ ] = pWord;
                  pWord = pSpace + 1;
            }

            /// Add rule for each token to sequence
          RuleItemArray ruleItemArray;
        SDRuleItem ruleItem;
          memset(&ruleItem, 0, sizeof(SDRuleItem));

        // Add StartOperationSequence item
        ruleItem.type=SD_RULE_STARTOPERATION;
        ruleItem.frequency= 0; // pPhnSpell->getFreq();
        ruleItem.hVoc= hScratchVoc;
        ruleItem.value.operation=SD_RULE_OPERATION_SEQUENCE;
        ruleItemArray.add(ruleItem);

            for (int i = 0; i < nTokens; i++ ) {
```

```
            CHK_SDAPI( phRules[i] = SDRule_GetHandle( hScratchVoc,
hScratchState, pTokenPtrs[i]) );

            if (phRules[i] == 0) {

          phRules[i] = spPhnSpellArray->getGuessRule(hScratchVoc,
hScratchState,

        pTokenPtrs[i], pTokenPtrs[i] );
                  // Add Rule item for next rule
              ruleItem.type=SD_RULE_RULE;
              ruleItem.frequency=0;
              ruleItem.hVoc= hScratchVoc;
              ruleItem.value.hRule = phRules[i];
              ruleItemArray.add(ruleItem);

                    // FUTURE: concatenate "bestPron" env vars
              }
          }

        // Add EndOperationSequence item
        ruleItem.type=SD_RULE_ENDOPERATION;
        ruleItem.frequency=0;
        ruleItem.hVoc= hScratchVoc;
        ruleItem.value.operation=SD_RULE_OPERATION_SEQUENCE;
        ruleItemArray.add(ruleItem);

        /// Add the new rule to the voc
        SDRuleItem *pRuleItems = ruleItemArray.getData();
        int nItems = ruleItemArray.count();
        CHK_SDAPI( SDhRule hNewRule= SDRule_New(hScratchVoc, hScratchState) );
        CHK_SDAPI( SDRule_SetDescription(hScratchVoc, hNewRule, pRuleItems,
nItems) );
        CHK_SDAPI( SDRule_SetName(hScratchVoc, hNewRule, szRuleName) );
          assert(hNewRule);

            // FUTURE: Set "bestPron" envvar

            /// Clean up
            DgnDelete(pTokenPtrs);
            DgnDelete(phRules);
            DgnDelete(pPhrase);
            return hNewRule;
      }
}


////////////////////////////////////////////////////////////////////////
//
// PronGuesser_GetRuleFromSpellingResult()
//
// Returns an SDhRule named pRuleName which contains rules for the first
// nSpellings-many results in hRes.
//
// hScratchState is the state in which to create the pron network of rules,
// states and words for guessing pronunciations.
//
// hSpellRes is the result of an utterance which used words in pSpellStateSpec
// to spell the word for which we will guess a pron.
//
// FUTURE Create a rule which contains sub rules as weighted alterntates.
SDhRule PronGuesser_GetRuleFromSpellingResult(SDhVoc hScratchVoc,
```

```
                                                        SDhState
hScratchState,
                                                const char *pRuleName,
                                                          SDhResult hSpellRes,
                                                SDStateSpec *pSpellStateSpec,
                                                    int nSpellings)
{
      if ( !spPhnSpellArray)
      errThrow( USE_ERR( PronGuesser, 1 ) );

   if ( hScratchVoc == 0 )
      errThrow( USE_ERR( PronGuesser, 2 ), hScratchVoc );

   if ( hSpellRes == 0 )
      errThrow( USE_ERR( PronGuesser, 2 ), hSpellRes );

   if ( pSpellStateSpec == NULL )
      errThrow( USE_ERR( PronGuesser, 3 ), "pSpellStateSpec" );


#if 0
      SDhState hNetworkState = SDState_New(hScratchVoc, hScratchState);
      SDState_SetName(hScratchVoc, hNetworkState, "Multi-spelling network
state");
#endif
      SDRuleItem ruleItem;
      ruleItem.type=SD_RULE_STATE;
      ruleItem.frequency=0;
      ruleItem.hVoc = hScratchVoc;
      ruleItem.value.hState=hScratchState;

      CHK_SDAPI( SDhRule hNetworkRule = SDRule_New(hScratchVoc, hScratchState)
);
      assert(hNetworkRule);
    CHK_SDAPI( SDRule_SetDescription(hScratchVoc, hNetworkRule, &ruleItem, 1)
);
    CHK_SDAPI( SDRule_SetName(hScratchVoc, hNetworkRule, pRuleName) );

      ChoiceList *pChList = DgnNew(ChoiceList);
   pChList->setConfiguration(0, 0, pSpellStateSpec->hVoc,
pSpellStateSpec->hState);
   pChList->init(hSpellRes);

   // Add the rule for the first character of each spelling to the network.
   int nSpell = pChList->getNEntries();
   if (nSpell > nSpellings)
         nSpell = nSpellings;
      while ( nSpell-- ) {
         SDhRule hNewRule = spPhnSpellArray->getGuessRule(hScratchVoc,

   hScratchState,

       pChList->getTranscript( nSpell ),

       pChList->getTranscript( nSpell ));

         assert( hNewRule );
         CHK_SDAPI( SDState_AddRule(hScratchVoc, hScratchState, hNewRule)
);
      }
//    memStats("Network Built");
    return hNetworkRule;
```

```
}


////////////////////////////////////////////////////////////////////////////
//
// PronGuesser_GetPronsFromResult()
//
// Returns the actual buffer length required to contain all the
pronunciations.
//
// PronGuesser_GetPronsFromResult will write up to nMaxProns-many
pronunciations
// into pPronBuf, in the order in which they are found in hResult.  hPronRule
is
// a pron network rule produced by PronGuesser_GetRuleFromSpellingResult() or
// PronGuesser_GetPronsFromResult() for the word for which we are guessing a
// pron.  hPronResult is from a recognition call in which the word in question

// was spoken and the grammar contained hPronRule in an appropriate manner.
//
// nMaxProns == -1 is a wildcard for all prons in hPronResult.
//
// If the buffer is not large enough, the pronunciations will be truncated.
// If the buffer is large enough, each pronunciations will be null-terminated
// and the final pronunciation will be double null-terminated.  All
// pronunciations will have length > 0.
//
// FUTURE use chlist module to process hResult
//
// If hPronResult is 0, then a single pron based on the pron guesser's
// internal language model will be written into pPronBuf.
//
size_t PronGuesser_GetPronsFromResult(const SDhVoc hVoc,
                                      const SDhRule hPronRule,
                                      const SDhResult hPronResult,
                                            const int nMaxProns,
                                            char *pPronBuf,
                                            const size_t lBuf)
{
      if ( !spPhnSpellArray )
      errThrow( USE_ERR( PronGuesser, 1 ) );

   if ( hPronRule == 0 )
      errThrow( USE_ERR( PronGuesser, 2 ), hPronRule, "hPronRule" );

      if ( pPronBuf == NULL )
      errThrow( USE_ERR( PronGuesser, 3 ), "pPronBuf" );

    if ( hPronResult == 0 ) { // return pron stored in env var "bestPron"

//      errThrow( USE_ERR( PronGuesser, 2 ), hPronResult, "hPronResult");

            CHK_SDAPI( SDhEnv hRuleEnv = SDRule_AccessEnv(hVoc, hPronRule,
SDENV_EXISTING) );
//          if (hRuleEnv == 0)
//                err
            CHK_SDAPI(int lPron =  SDEnv_GetData(hRuleEnv, "bestPron",
pPronBuf, lBuf) );
            return lPron;
      }
```

```
#if 1
      ChoiceList *pChList = DgnNew(ChoiceList);
      pChList->setConfiguration(hVoc, hPronRule, 0, 0);

      pChList->setConfiguration( 1, 1, 1, 1, 0);
      pChList->init(hPronResult);


      int nProns = pChList->getNEntries();
    if (nProns == 0)
      return 0;

      if (nProns > nMaxProns) {
            nProns = nMaxProns;
      }

      int nPronsFound = 0;
      size_t totalBufSizeNeeded = 0;
      size_t nBufSize = lBuf;
      char *pBuf = &pPronBuf[0];
      memset(pBuf, 0, nBufSize);
      size_t lenNewPron = 0;
      const char *pNewPron = NULL;

      for ( int i = 0; (pNewPron = pChList->getPron(i)) != NULL ; i++ ) {
            if (nPronsFound == nProns){
                  break;
            }

            if ( pNewPron[0] == 0 ){
                  continue;
            }

            nPronsFound++;
            if (pNewPron[0] == '_')
            { // Skip first phoneme if it is silence
                  pNewPron++;
            }

            lenNewPron = strlen(pNewPron) + 1;
            // update total size, including prons we don't have room for.
            totalBufSizeNeeded += ( lenNewPron );

            // output buffer large enough ?
            if (lenNewPron <= nBufSize) {
                  // append new pron (w/one Null) to the output buffer
                  strncpy(pBuf, pNewPron, nBufSize);
                  pBuf += (lenNewPron);
                  nBufSize -= (lenNewPron);
            } else {
                  pBuf = 0;
                  nBufSize = 0;
            }
      }
      assert(nPronsFound);

      if (totalBufSizeNeeded >= lBuf) {
            pPronBuf[lBuf] = 0;
      } else {
            // finish the pron(s) by adding the extra 0
            pPronBuf[totalBufSizeNeeded] = 0;
            totalBufSizeNeeded++;
```

```
        }
        DgnDelete(pChList);
        return totalBufSizeNeeded;
}
#else

        SDResultInfo resultInfo;
        CHK_SDAPI( SDResult_GetInfo(hPronResult, &resultInfo) );

        int nResTokens =  128;
        SDResultToken *pResTokenBuf = DgnNew( SDResultToken[ nResTokens ] );

    if (resultInfo.nChoices == 0)
        return 0;

        int nPronsFound = 0;
        size_t totalBufSizeNeeded = 0;
        size_t nBufSize = lBuf;
        char *pBuf = &pPronBuf[0];
        memset(pBuf, 0, nBufSize);
        size_t retSize = 0;

        int bCopyPron = 0;

        SDResultChoiceInfo resChoiceInfo;

        // Start looping over the entries in the choice list
        int rank;
        for( rank = 0 ; rank < resultInfo.nChoices ; ++rank ) {

              // Set up Choice Token buffer
              memset(pResTokenBuf, 0, nResTokens);
              CHK_SDAPI( int nTokens = SDResult_GetChoiceTokens( hPronResult,
rank, pResTokenBuf, nResTokens ) );
              if (nTokens > nResTokens) {
                    DgnDeleteArray(pResTokenBuf);
                    pResTokenBuf = DgnNew(SDResultToken[ nTokens ]);
                    memset(pResTokenBuf, 0, nResTokens);
                    CHK_SDAPI( nResTokens = SDResult_GetChoiceTokens(
hPronResult, rank, pResTokenBuf, nTokens) );
              }
              CHK_SDAPI( SDResult_GetChoiceInfo(hPronResult, rank,
&resChoiceInfo) );
              CHK_SDAPI( SDResultToken *pRes = &pResTokenBuf[0]  );

              // Now parse the result token buffer and extract a pron from the
              // sub-path between STARTRULE(hPronRule) and ENDRULE(hPronRule)
              int entry = 0;
        int foundPron = 0;
              while( entry < nTokens ){
                    switch( pRes->type ) {
                     case SD_RESULT_STARTRULE:
                          if (hPronRule == pRes->value.rule.hRule    &&
                                hVoc == pRes->value.rule.hVoc) {
                                assert(bCopyPron == 0);
                                bCopyPron = 1;
                          }
                          break ;

                     case SD_RESULT_ENDRULE:
                          if (hPronRule == pRes->value.rule.hRule &&
                                hVoc == pRes->value.rule.hVoc)      {
```

```
                              assert(bCopyPron == 1);
                              bCopyPron = 0;
                              foundPron = 1;
                        }
                        break ;

                  case SD_RESULT_WORD:
                        if (bCopyPron)    {
                              // Convert a continuous recognition on fragments
to a pron for a word
                              CHK_SDAPI( retSize = SDWord_GetPronunciations(
pRes->value.word.hVoc,

      pRes->value.word.hWord,

      (unsigned char *)pBuf, nBufSize) );
                              totalBufSizeNeeded += (retSize - 2);
                              if (retSize <= nBufSize) {
                                    pBuf += (retSize - 2);
                                    nBufSize -= (retSize - 2);
                              } else {
                                    pBuf = 0;
                                    nBufSize = 0;
                              }
                        }
                        break ;
                  }
         if (foundPron)
            break;
                  pRes++;
                  entry++;
            }
            assert(bCopyPron == 0);
      if (foundPron)
      {
            pPronBuf[totalBufSizeNeeded] = 0;
         pBuf++;
         nBufSize--;
            totalBufSizeNeeded++;
         nPronsFound++;
      }
            if (nMaxProns >= 0 && nPronsFound >= nMaxProns)
         break;
      } // end loop on choices

      assert(nPronsFound);

      if (totalBufSizeNeeded >= lBuf) {
            pPronBuf[lBuf] = 0;
      } else {
            // finish the pron(s) by adding the extra 0
            pPronBuf[totalBufSizeNeeded] = 0;
            totalBufSizeNeeded++;
      }
      DgnDeleteArray(pResTokenBuf);
      return totalBufSizeNeeded;
}
#endif

void PronGuesser_DeleteValidationState(SDhVoc hVoc, SDhState hState)
{
      // How many words in test State?
```

```
      SDStateInfo validStateInfo;
      CHK_SDAPI( SDState_GetInfo(hVoc, hState, &validStateInfo) );

      // We want to remove the state and all its words from the voc,
      if (validStateInfo.nWords) {

            // Get and fill a buffer with their handles
            SDhWord *phWordBuf = DgnNew( SDhWord[ validStateInfo.nWords ] );
            CHK_SDAPI( SDhWordIterator hWIter =
                  SDState_IterateWords( hVoc, hState,
                                    SDHCOLL_NOCOLLATION, SD_WORD_NORESTRICTION,
"") );
            assert(hWIter);
            CHK_SDAPI( SDInteger nGotWords = SDWord_NextGroup( phWordBuf,
                                          validStateInfo.nWords, hWIter )
);
            assert(nGotWords == validStateInfo.nWords);
            // Take them out of the test State & Voc
            while(nGotWords--) {
                  CHK_SDAPI( SDState_DeleteWord( hVoc, hState,
phWordBuf[nGotWords] ) );
                  CHK_SDAPI( SDWord_Delete( hVoc, phWordBuf[nGotWords] ) );
            }
            CHK_SDAPI( SDWord_EndIteration( hWIter ) );
            if (phWordBuf)
                  DgnDeleteArray( phWordBuf );
      }
      CHK_SDAPI( SDState_Delete( hVoc, hState ) );

}


size_t PronGuesser_GetValidProns(SDhVoc hVoc, SDhState hValidState,
                                          SDhResult hPronResult, int
nMaxProns,
                                          char *pPronBuf, size_t lBuf)
{
      if ( !spPhnSpellArray )
      errThrow( USE_ERR( PronGuesser, 1 ) );

   if ( hValidState == 0 )
      errThrow( USE_ERR( PronGuesser, 2 ), hValidState );

   if ( hPronResult == 0 )
      errThrow( USE_ERR( PronGuesser, 2 ), hPronResult);

      if ( pPronBuf == NULL )
      errThrow( USE_ERR( PronGuesser, 3 ), "pPronBuf" );

      ChoiceList *pCL = DgnNew(ChoiceList);
      pCL->setConfiguration( 0,0, hVoc, hValidState );
      pCL->setConfiguration( 1,1,1,1,0 );
      pCL->init(hPronResult);
      size_t totalSize = 0;
      int nProns =0;
   for ( int choiceNum = 0; choiceNum < pCL->getNEntries(); choiceNum++ )
   {
      const char *pPron = pCL->getPron(choiceNum);
      if ( pPron[0] == 0 )
         break;

        while (pPron[0] == '_')
```

```
        { // Skip first phoneme if it is silence
            pPron++;
        }

      size_t lPron = strlen(pPron);
      if ( totalSize + lPron < lBuf )
      {
         if (nProns == 0)
            strcpy(pPronBuf, pPron);
         else
            strscat( pPronBuf, pPron );
         nProns++;
      }
      totalSize += lPron;
      if ( nProns >= nMaxProns )
         break;
   }
   DgnDelete( pCL );
   return totalSize;
}


SDhState PronGuesser_CreateValidationState(SDStateSpec *pValidationStateSpec,
                                                                    char
*pPronBuf)
{
   // Create a state in the validation vocabulary for testing
   CHK_SDAPI( SDhState hPronTestState =
SDState_New(pValidationStateSpec->hVoc, 0) );
   assert(hPronTestState);
      CHK_SDAPI( SDState_SetName(pValidationStateSpec->hVoc,
hPronTestState,
                                 "Pron Candidate State") );
   CHK_SDAPI( SDState_SetLMAllowed(pValidationStateSpec->hVoc, hPronTestState,
1) );

   CHK_SDAPI( SDState_AddState(pValidationStateSpec->hVoc,
                          pValidationStateSpec->hState,
                                hPronTestState) );

      int nDupProns = 0;
      int nAdded = 0;
      char tmpPronBuf[500];
      char *pPron = pPronBuf;
   while(*pPron)
   {
         char *pTmpPron = tmpPronBuf;
            memset(tmpPronBuf, 0, 500);
            while ( (*pTmpPron = *pPron) != 0)
            {
                  pTmpPron++;
                  pPron++;
            }
            pPron++;
      *pTmpPron = ' ';    // Avoid collision with existing words in Voc
      // pTmpPron++;
            assert( pTmpPron - tmpPronBuf < 500-2 );

         CHK_SDAPI( SDhWord hNewWord =
SDWord_GetHandle(pValidationStateSpec->hVoc, tmpPronBuf) );
         if (hNewWord)
      {
```

```
#ifndef SHIP
                xprintf("pron # %d '%s' duplicates id # %d\n", nAdded +
nDupProns, tmpPronBuf, hNewWord);
#endif
                nDupProns++;
            }
        else
        {
                hNewWord = SDWord_New(pValidationStateSpec->hVoc, tmpPronBuf);
           *pTmpPron = 0;    // Avoid collision with existing words in Voc
              SDWord_SetPronunciations(pValidationStateSpec->hVoc, hNewWord,
                                                                (unsigned char *)
tmpPronBuf) ;
                    SDState_AddWord(pValidationStateSpec->hVoc,
                                hPronTestState, hNewWord);
              nAdded++;
        }
    }
        xprintf("Created %d candidate words, Skipped %d duplicate prons\n",
nAdded, nDupProns);


        if ( !nAdded )
    {
                xprintf("No candidate prons for validation\n");
        }
        return hPronTestState;
}

////////////////////////////////////////////////////////////////////////
//
// PronGuesser_DumpScratchState
// Write the "pron-network" produced by  PronGuesser_GetRuleFromString() and
// PronGuesser_GetRuleFromSpellingResult() using xprintfs.
//
void PronGuesser_DumpScratchState(SDhVoc hScratchVoc, SDhState hScratchState)
{
    if (hScratchVoc == 0)
        errThrow( USE_ERR( PronGuesser, 2), "hScratchVoc" );

    if (hScratchState == 0)
        errThrow( USE_ERR( PronGuesser, 2), "hScratchState" );

        // Dump all child rules
     CHK_SDAPI( SDhRuleIterator hRuleIter =
SDState_IterateChildRules(hScratchVoc,

                        hScratchState) );
        SDhRule hRule = 0;
     showErrorAndReset( PREV_ERR, __FILE__, __LINE__, "" );
     while( (hRule = SDRule_Next( hRuleIter )) != 0 )
     {
         showErrorAndReset( PREV_ERR, __FILE__, __LINE__, "SDRule_Next()" );
             xDumpRule(hScratchVoc, hRule);
     }
        CHK_SDAPI( SDRule_EndIteration( hRuleIter ) );

        // Dump all child states
        CHK_SDAPI( SDhStateIterator hStateIter =
SDState_IterateChildren(hScratchVoc,

          hScratchState) );
```

```
      SDhState hState = 0;
    showErrorAndReset( PREV_ERR, __FILE__, __LINE__, "" );
      while( (hState = SDState_Next( hStateIter )) != 0 )
      {
        showErrorAndReset( PREV_ERR, __FILE__, __LINE__, "SDRule_Next()" );
            xDumpState(hScratchVoc, hState);
      }
      CHK_SDAPI( SDState_EndIteration( hStateIter ) );
} // PronGuesser_Dump...


////////////////////////////////////////////////////////////////////////////
//
// PronGuesser_CleanUpScratchState
//
// Clean up after PronGuesser_GetRuleFromString() and
// PronGuesser_GetRuleFromSpellingResult().
//
// PronGuesser_CleanUpScratchState deletes all the child rules and child
// states from hScratchState, as well as the words in the child states.
// This invalidates any existing pron network rule which were built with
// hScratchState.
//
// Note: This removes all the pron-networks in hScratchState, but not the
// factory which builds them.  To delete the factory, use
PronGuesser_Terminate()
//
void PronGuesser_CleanUpScratchState(SDhVoc hScratchVoc, SDhState
hScratchState)
{
   if (hScratchVoc == 0)
      errThrow( USE_ERR( PronGuesser, 2), "hScratchVoc" );

   if (hScratchState == 0)
      errThrow( USE_ERR( PronGuesser, 2), "hScratchState" );

      SDStateInfo stateInfo;

      // Remove all child rules
      // First we use an iterator to fill an array with their handles
 //   SDState_GetInfo(hScratchVoc, hScratchState, &stateInfo);
//    if (stateInfo.nChildRules)
//    {
    DgnAC< SDhRule > pRuleAC;
//          SDhRule *phRuleArray = DgnNew( SDhRule[ stateInfo.nChildRules ]);
          CHK_SDAPI( SDhRuleIterator hRuleIter =
SDState_IterateChildRules(hScratchVoc, hScratchState) );
          SDhRule hRule = 0;
        showErrorAndReset( PREV_ERR, __FILE__, __LINE__, "" );
          while( (hRule = SDRule_Next( hRuleIter )) != 0 ) {
            showErrorAndReset( PREV_ERR, __FILE__, __LINE__, "SDRule_Next()"
);
            //    phRuleArray[nRules++] = hRule;
            pRuleAC.add(hRule);
          }
//          assert(nRules == stateInfo.nChildRules);
            CHK_SDAPI( SDRule_EndIteration( hRuleIter ) );
            // Now that we are done with the iterator, we can kill off the
rules w/o
            // messing up the iteration
            int nRules = pRuleAC.getCount();
```

```
            while( nRules-- ) {
                  CHK_SDAPI( SDRule_Delete( hScratchVoc, pRuleAC[ nRules ] )
);
            }

//        if ( pRuleAC ) {
//              DgnDelete(pRuleAC);
  //      }
//    }

      // Remove all child states
      CHK_SDAPI( SDState_GetInfo(hScratchVoc, hScratchState, &stateInfo) );
      if (stateInfo.nChildStates)
      {
            SDhState *phStateArray = DgnNew( SDhState[ stateInfo.nChildStates
]);
      CHK_SDAPI( SDhStateIterator hStateIter =
SDState_IterateChildren(hScratchVoc, hScratchState) );
         SDhState hState = 0;
            int nStates = 0;
            int nChildStates = stateInfo.nChildStates;
       showErrorAndReset( PREV_ERR, __FILE__, __LINE__, "" );
         while( (hState = SDState_Next( hStateIter )) != 0 )
         {
            showErrorAndReset( PREV_ERR, __FILE__, __LINE__, "SDState_Next()"
);
                  phStateArray[nStates++] = hState;
                  // Remove all the words from the state
                  CHK_SDAPI( SDState_GetInfo(hScratchVoc, hState, &stateInfo)
);
                  if (stateInfo.nWords)
                  {
                        // Get and fill a buffer with their handles
                        SDhWord *phWordBuf = DgnNew( SDhWord[ stateInfo.nWords
] );
                        CHK_SDAPI( SDhWordIterator hWIter =
                              SDState_IterateWords( hScratchVoc, hState,
SDHCOLL_NOCOLLATION,

SD_WORD_NORESTRICTION, "") );
                        assert(hWIter);
                        CHK_SDAPI( SDInteger nGotWords =
                              SDWord_NextGroup(phWordBuf, stateInfo.nWords,
hWIter) );
                        assert(nGotWords == stateInfo.nWords);

                        // Take them out of the test State & Voc
                        while( nGotWords-- ) {
                              CHK_SDAPI( SDState_DeleteWord(hScratchVoc,
hState, phWordBuf[nGotWords]) );
                              CHK_SDAPI( SDWord_Delete(hScratchVoc,
phWordBuf[nGotWords]) );
                        }
                        CHK_SDAPI( SDWord_EndIteration(hWIter) );
                        if (phWordBuf)
                              delete [] phWordBuf;
                  }
      }
            assert(nStates == nChildStates);
            CHK_SDAPI( SDState_EndIteration( hStateIter ) );
            // Now that we are done with the iterator, we can kill off the
States w/o
```

```
            // messing up the iteration
            if ( nStates )
            {
                  while( nStates-- )
                  {
                        CHK_SDAPI( SDState_Delete( hScratchVoc, phStateArray[
nStates ] ) );
                  }
            }
      if (phStateArray)
                  DgnDeleteArray(phStateArray);
      }
}

/* old Trec history follows:
    8     11/07/96 12:11 Chuck
    TAHITI Ver 0.04.390
    Now support Silence in pron guessing, if it shows up in SDResult.
    Support for Silence within pron guesses, but not at front of pron.

    7     10/07/96 11:39 Chuck
    TAHITI Ver 0.04.337
    Reactivated pron-network dumping after validation changes.

    6     9/30/96 6:36p Joel
    TAHITI Ver 0.04.317

    5     9/13/96 9:41a Chuck
    TAHITI Ver 0.04.307
    New validation scheme for prons which splits apart
    PronGuesser_ValidateProns() and allows caller to do the recog call.

    4     8/06/96 6:24p Chuck
    TAHITI Ver 0.04.252
     Added reporting of pron for built and dict words

    3     7/29/96 10:19a Joel
    TAHITI Ver 0.04.233

    2     7/23/96 2:44p Chuck
    TAHITI Ver 0.04.228
     Added docs for PronGuesser interface.
     We now use THROW_ERR instead of assertions.

    1     7/17/96 2:32p Chuck
    TAHITI Ver 0.04.210
     This is a C-function wrapper which uses PhnSpellArray class to do pron
    guessing
    $NoKeywords: $
*/
```

```
/*//////////////////////////////////////////////////////////////////////////

  FILE:                  prnguess.h
  CREATED:
  AUTHOR:          Chuck Ingold
  DESCRIPTION:

    Copyright (C) Dragon Systems, 1995-1996
    DRAGON SYSTEMS CONFIDENTIAL

      VSS revision history.  Do not edit by hand.
    $Log: /pq/prons/prnguess.h $

    1     3/24/97 16:30 Chuck
    PHONEQUERY Ver 0.01.165
    Added prons lib

    $NoKeywords: $

*/ //////////////////////////////////////////////////////////////////////////

#ifndef _prnguess_h_
#define _prnguess_h_


//#include "sdapi.h"
//#include "phnspell.h"

// PronGuess.h
//
// This module contains a pronunciation guesser.  Pronunciation guessing
// works as follows:
//
// 1) Initialize a "pron-network factory" by calling PronGuesser_LoadAscii()
//    or PronGuesser_Load().
//
// 2) To guess a pronunciation for a word, create a pron-network for the
//    word by calling either PronGuesser_GetRuleFromString() or
//    PronGuesser_GetRuleFromSpellingResult().
//
// 3) Insert the resulting rule in an FSG grammer and call recognition.
//
// 4) Extract the prons By calling PronGuesser_GetPronsFromResult().
//
// 5) (Optional) Write the pron-network to a log file for debugging by calling

//    PronGuesser_DumpScratchState().
//
// 6) (Optional) Remove the pron-network of SDAPI rules, states and words by
//    calling PronGuesser_CleanUpScratchState().
//
// 7) (Optional) Validate the prons as follows:
// 7a) Prepare by calling PronGuesser_CreateValidationState().
// 7b) Extract the valid prons by calling PronGuesser_GetValidProns().
// 7c) Remove the validation state by calling
PronGuesser_DeleteValidationState().
//
// 8) Assign the prons to an SDhWord by calling SDWord_SetPronunciations().
//
// 9)(Optional) Shut down the "pron-network factory" by calling
//    PronGuesser_Terminate().
//
```

```
// FUTURE Add PronGuesser handles for different PhnSpellArrays or algorithms
//        such as phonetic recognizer


//   Initializes the internal PronGuesser data from an ascii file.
void PronGuesser_LoadAscii(const char *szFileName);

//   Initializes the internal PronGuesser data from a binary file.
void PronGuesser_Load(FILE *pFile);

// Writes PronGuesser internal data to a binary file.
void PronGuesser_Save(FILE *pFile);

// Deletes internal PronGuesser data.
//
// Note: This removes the factory, not the rules, states and words which
// make up a pron-network composed of SDAPI objects.  To delete pron-networks,
// use PronGuesser_CleanUpScratchState()
void PronGuesser_Terminate();

// Returns an SDhRule named pRuleName which contains a pron-network
// for szSpelling.
//
// hScratchVoc and hScratchState specify where to create the network of rules,
// states and words for guessing pronunciations.
//
SDhRule PronGuesser_GetRuleFromString(SDhVoc hScratchVoc,
                                      SDhState
hScratchState,
                                      const char
*szRuleName,
                                      const char
*szSpelling);

// Returns an SDhRule named pRuleName which contains pron-network rules for
// the first nSpellings-many results in hSpellResult.
//
// hScratchVoc and hScratchState specify where to create the network of rules,
// states and words for guessing pronunciations.
//
// hSpellRes is the result of an utterance which used words in pSpellStateSpec
// to spell the word for which we will guess a pron.
//
SDhRule PronGuesser_GetRuleFromSpellingResult(SDhVoc hScratchVoc,
                                              SDhState hParentState,
                                      const char
*szRuleName,
                                                    SDhResult
hSpellResult,
                                      SDStateSpec
*pSpellStateSpec,
                                                    int
nSpellings);

// Returns the actual buffer length required to contain all the
pronunciations.
//
// PronGuesser_GetPronsFromResult will write up to nMaxProns-many
pronunciations
// into pPronBuf, in the order in which they are found in hResult.

//  hPronRule is
```

```
// a pron network rule produced by PronGuesser_GetRuleFromSpellingResult()
// or PronGuesser_GetPronsFromResult() for the word for which we are guessing
a
// pron.  hPronResult is from a recognition call in which the word in question
// was spoken and the grammar contained hPronRule in an appropraite manner.
//
// If the buffer is not large enough, the pronunciations will be truncated.
// If the buffer is large enough, each pronunciations will be null-terminated
// and the final pronunciation will be double null-terminated.  All
// pronunciations will have length > 0.
size_t PronGuesser_GetPronsFromResult(const SDhVoc hRuleVoc,
                  const SDhRule hPronRule, const SDhResult hPronResult,
                  const int nMaxProns, char *pPronBuf, const size_t lBuf);

// Creates a validation state in the same voc as pValidationStateSpec->hVoc,
// and populates it with words made out of the prons in pPronBuf.
SDhState PronGuesser_CreateValidationState(SDStateSpec *pValidationStateSpec,
                                                    char *pPronBuf);

// Returns the actual buffer length required to contain all the
pronunciations.
//
// PronGuesser_GetPronsFromResult will write up to nMaxProns-many
pronunciations
// into pPronBuf, in the order in which they are found in hResult.
//  hPronRule is
// a pron network rule produced by PronGuesser_GetRuleFromSpellingResult()
// or PronGuesser_GetPronsFromResult() for the word for which we are guessing
a
// pron.  hPronResult is from a recognition call in which the word in question
// was spoken and the grammar contained hPronRule in an appropraite manner.
//
// If the buffer is not large enough, the pronunciations will be truncated.
// If the buffer is large enough, each pronunciations will be null-terminated
// and the final pronunciation will be double null-terminated.  All
// pronunciations will have length > 0.
size_t PronGuesser_GetValidProns(SDhVoc hVoc, SDhState hValidState,
                                                SDhResult hPronResult, int
nMaxProns,
                                                char *pPronBuf, size_t lBuf);

// Removes the validation state and its contents.
void PronGuesser_DeleteValidationState(SDhVoc hVoc, SDhState hState);

// Writes the "pron-network" produced by  PronGuesser_GetRuleFromString() and
// PronGuesser_GetRuleFromSpellingResult() using xprintfs.
void PronGuesser_DumpScratchState(SDhVoc hScratchVoc, SDhState hScratchState);

// Clean up after PronGuesser_GetRuleFromString() and
// PronGuesser_GetRuleFromSpellingResult().
//
// PronGuesser_CleanUpScratchState deletes all the child rules and child
// states from hScratchState, as well as the words in the child states.
// This invalidates any existing pron-network rule(s) built with
hScratchState.
//
// Note: This removes all the pron-networks in hScratchState, but not the
// factory which builds them. To delete the factory, use
PronGuesser_Terminate()
void PronGuesser_CleanUpScratchState(SDhVoc hScratchVoc, SDhState
hScratchState);
```

```
/* Old TREC history follows:

    5     10/07/96 11:39 Chuck
    TAHITI Ver 0.04.337
    Reactivated pron-network dumping after validation changes.

    4     9/13/96 9:41a Chuck
    TAHITI Ver 0.04.307
    New validation scheme for prons which splits apart
    PronGuesser_ValidateProns() and allows caller to do the recog call.
    Removed extraneous "trec.h"

    3     7/29/96 10:19a Joel
    TAHITI Ver 0.04.233

    2     7/23/96 2:44p Chuck
    TAHITI Ver 0.04.228
     Added docs for PronGuesser interface.
     We now use THROW_ERR instead of assertions.

    1     7/17/96 2:32p Chuck
    TAHITI Ver 0.04.210
     This is a C-function wrapper which uses PhnSpellArray class to do pron
    guessing

*////////////////////////////////////////////////////////////////////////////

#endif
```

d hist

```
     (FILE 'USPAT' ENTERED AT 08:07:43 ON 30 JAN 1999)
L1          1762 S TWO (2A) (LETTER OR LETTERS)
L2             1 S 4718094/PN
L3             1 S L1 AND L2
L4           243 S CLASS (3W) LIST
L5             5 S L1 AND L4
L6        339317 S WORD OR WORDS
L7           172 S L4 AND L6
L8             5 S L7 AND L1
L9          1871 S PHONETIC OR PHONEM?
L10            1 S L8 AND L9
                 SET HIGH OFF
L11            1 S L10 AND L10
                 SET HIGH ON
L12            1 S L1 AND L11
L13            1 S L12 AND L4
L14            1 S L13 AND L9
```

=> d kwic

US PAT NO: 4,471,459 [IMAGE AVAILABLE] L14: 1 of 1

SUMMARY:

BSUM(21)

Another approach discussed in that article involves the frequency of **two letter** pairs and three letter triples to detect potential misspellings in order to form an index into a table of acceptable. . .

SUMMARY:

BSUM(22)

Other . . . any type of automatic matching of misspelled words. Another technique employed is to take tokens and convert them into standard **phonetic** spelling and to find similar sounding words in a dictionary. This, for example, works well with double errors using, for. . .

DETDESC:

DETD(55)

The . . . of the textual data base is a dictionary of entry words which are stored and are accessible by the first **two letters**. All of the words having the same first **two letters** are stored together. For example, representations of significant words beginning with the letters AA are arranged together, representations of significant. . .

DETDESC:

DETD(66)

The . . . words) i[redacted]rranged, stored and accessible [redacted] the disk storage device 1107, a[redacted] called a secondary storage, b[redacted]he first **two letters** of the word (i.e., in "families"). By calling a data base services program, the program QFLPKG obtains the family of. . . variable length character stem. Using the query word "HELPS" as an example, all data base entry words having the first **two letters** HE are put into the ENTRIES buffer of RAM 1104 for processing against the query word.

DETDESC:

DETD(127)

The . . . in place of the actual suffixes, places the corresponding suffix indication. All of the resultant suffix indications for each suffix **class** indication in the **list** then become pointers to the rows of the SUFFIX.sub.-- TABLE 1204 where the actual suffixes can be located and read. . .

DETDESC:

DETD(134)

The . . . returned entry words is determined using the size value (SIZE) for the stem of the query word and the misspelling **class** indication. The **list** of acceptable suffixes is then compared with the suffixes determined in each of the returned entry words and equality is. . .

DETDESC:

DETD(163)

A . . . query word from among the family of significant entry words of the data base which begin with the same first **two letters** as the query word. It is to be noted that the invention is not limited to requirements for a match. . .

DETDESC:

DETD(171)

The . . . is a pointer to the location in external RAM 1104 where the family of entry words, beginning with the same **two letters** as the query word, is located. NUMENT is a word value giving the number of entry words in the RAM. . .

DETDESC:

DETD(189)

Consider . . . word HELP (by way of example, HEBREW, HELP, HELPS, and HEPLS), that is, all words beginning with the same first **two letters** as the query word HELP. It should also be noted that the number of entries in buffer 1402 is indicated. . .

```
 s speech (10a) recogni?

         15602 SPEECH
        305303 RECOGNI?
L1        2646 SPEECH (10A) RECOGNI?

=> s spell?

L2        3482 SPELL?

=> s l1 and l2

L3         214 L1 AND L2

=> s phonem?

L4         948 PHONEM?

=> s l3 and l4

L5          99 L3 AND L4

=> s rules

L6       27943 RULES

=> s l5 and l6

L7          52 L5 AND L6

=> s string?

L8       87908 STRING?

=> s l7 and l8

L9          37 L7 AND L8

=> s phonetic pronunciation?

           934 PHONETIC
           769 PRONUNCIATION?
L10         22 PHONETIC PRONUNCIATION?
                 (PHONETIC(W)PRONUNCIATION?)

=> s l9 and l10

L11          0 L9 AND L10

=> s utterance

L12        780 UTTERANCE

=> s l10 and l12

L13          4 L10 AND L12

=> s l9 and l12
```

L14 19 L9 AND L[redacted]

=> s l13 and l14

L15 0 L13 AND L14

=> d l13 1-

1. 5,737,490, Apr. 7, 1998, Method and apparatus for constructing continuous parameter fenonic hidden markov models by replacing phonetic models with continous fenonic models; Stephen Christopher Austin, et al., 1/1 [IMAGE AVAILABLE]

2. 5,521,324, May 28, 1996, Automated musical accompaniment with multiple input sensors; Roger B. Dannenberg, et al., 84/612, 613, 631, 633, DIG.4 [IMAGE AVAILABLE]

3. 5,384,893, Jan. 24, 1995, Method and apparatus for speech synthesis based on prosodic analysis; Sandra E. Hutchins, 704/267 [IMAGE AVAILABLE]

4. 3,603,738, Sep. 7, 1971, TIME-DOMAIN PITCH DETECTOR AND CIRCUITS FOR EXTRACTING A SIGNAL REPRESENTATIVE OF PITCH-PULSE SPACING REGULARITY IN A SPEECH WAVE; Louis R. Focht, 704/207 [IMAGE AVAILABLE]

=> d l14 1-

1. 5,732,395, Mar. 24, 1998, Methods for controlling the generation of speech from text representing names and addresses; Kim Ernest Alexander Silverman, 704/260, 258, 266, 267 [IMAGE AVAILABLE]

2. 5,719,997, Feb. 17, 1998, Large vocabulary connected **speech recognition** system and method of language representation using evolutional grammer to represent context free grammars; Michael Kenneth Brown, et al., 1/1 [IMAGE AVAILABLE]

3. 5,699,456, Dec. 16, 1997, Large vocabulary connected **speech recognition** system and method of language representation using evolutional grammar to represent context free grammars; Michael Kenneth Brown, et al., 382/226, 190 [IMAGE AVAILABLE]

4. 5,682,501, Oct. 28, 1997, Speech synthesis system; Richard Anthony Sharman, 704/260, 256, 257, 258, 261, 266, 269 [IMAGE AVAILABLE]

5. 5,652,828, Jul. 29, 1997, Automated voice synthesis employing enhanced prosodic treatment of text, **spelling** of text and rate of annunciation; Kim Ernest Alexander Silverman, 704/260, 258, 266, 267 [IMAGE AVAILABLE]

6. 5,638,425, Jun. 10, 1997, Automated directory assistance system using word recognition and **phoneme** processing method; Frank E. Meador, III, et al., 379/88, 89, 201; 704/231, 236, 251, 270 [IMAGE AVAILABLE]

7. 5,623,609, Apr. 22, 1997, Computer system and computer-implemented process for phonology-based automatic **speech recognition**; Jonathan Kaye, et al., 704/1, 231, 255 [IMAGE AVAILABLE]

8. 5,293,584, Mar. 8, 1994, **Speech recognition** system for natural language translation; Peter F. Brown, et al., 704/277 [IMAGE AVAILABLE]

9. 5,293,451, Mar. 8, 1994, Method and apparatus for generating models of spoken words based on a small number of utterances; Peter F. Brown, et al., 704/245 [IMAGE AVAILABLE]

10. 5,222,188, Jun. 2[redacted]1993, Method and apparatus for [redacted]eech **recognition** based on subsyllable **spellings**; Sandra E. Hutchins, 704/200 [IMAGE AVAILABLE]

11. 5,208,897, May 4, 1993, Method and apparatus for **speech recognition** based on subsyllable **spellings**; Sandra E. Hutchins, 704/200 [IMAGE AVAILABLE]

12. 5,091,950, Feb. 25, 1992, Arabic language translating device with pronunciation capability using language pronunciation **rules**; Moustafa E. Ahmed, 704/277 [IMAGE AVAILABLE]

13. 5,075,896, Dec. 24, 1991, Character and **phoneme** recognition based on probability clustering; Lynn D. Wilcox, et al., 382/225, 228 [IMAGE AVAILABLE]

14. 5,054,074, Oct. 1, 1991, Optimized **speech recognition** system and method; Raimo Bakis, 704/240 [IMAGE AVAILABLE]

15. 5,027,406, Jun. 25, 1991, Method for interactive **speech recognition** and training; Jed Roberts, et al., 704/244 [IMAGE AVAILABLE]

16. 4,884,972, Dec. 5, 1989, Speech synchronized animation; Elon Gasper, 434/185; 345/302, 473; 434/167, 169, 307R; 704/235 [IMAGE AVAILABLE]

17. 4,852,170, Jul. 25, 1989, Real time computer **speech recognition** system; Theodore A. Bordeaux, 704/277 [IMAGE AVAILABLE]

18. 4,489,435, Dec. 18, 1984, Method and apparatus for continuous word **string** recognition; Stephen L. Moshier, 704/244 [IMAGE AVAILABLE]

19. 4,481,593, Nov. 6, 1984, Continuous **speech recognition**; Lawrence G. Bahler, 704/253 [IMAGE AVAILABLE]